Une utilisation du modèle MVC pour une plate-forme de travail virtuel

Nicolas Martin
Nicolas.Martin@lifl.fr

Samuel Degrande
Samuel.Degrande@lifl.fr

Christophe Chaillou
Christophe.Chaillou@lifl.fr

Laboratoire d'Informatique Fondamentale de Lille
Bâtiment M3, Cité Scientifique
59655, Villneuve d'Ascq, France

RESUME
De nos jours, les interfaces 3D apparaissent de plus en plus dans les environnements utilisateurs. L'arrivée de la troisième dimension apporte certaines améliorations aux interfaces mais les différents systèmes existant ne font qu'intégrer des applications 2D dans un environnement virtuel sans proposer de réelles nouveautés en terme d'Interface Homme-Machine 3D. Dans cette article, nous effectuons un retour d'expérience sur l'utilisation du modèle MVC pour l'architecture d'un bureau 3D collaboratif, afin d'offrir une plate-forme de développement totalement ouverte : Spin|3D.

MOTS CLES : Bureau virtuel 3D, MVC, Architecture logicielle, IHM 3D.

ABSTRACT
Nowadays, 3D interfaces appear more and more in the users' environments. The arrival of the third dimension adds certain contributions to the interfaces but the existing systems only integrate 2d applications in a virtual environment without proposing real innovation in term of chi 3d. In this article, we carry out a feedback on the use of the MVC paradigm for the architecture of a 3d desktop: Spin|3D.

CATEGORIES AND SUBJECT DESCRIPTORS: D.2.11 [Software Engineering]: Software Architectures; H.5.2 [User Interfaces]: User interface management systems (UIMS).

GENERAL TERMS: Design

KEYWORDS: 3D Virtual Desktop, MVC, software architecture, 3D CHI.

INTRODUCTION

Les interfaces de travail utilisateurs s'appuient toujours en majorité sur la métaphore du bureau 2D introduite par Xerox Star et popularisée sur Macintosh il y a 20 ans. On voit tout de même apparaître des effets visuels tels que l'ombre et la transparence, introduites par les interfaces 2D 1/2. Mais les gestionnaires d'environnement n'évoluent globalement pas.

Les ordinateurs grand public sont devenus de plus en plus puissants et permettent l'exécution d'applications virtuelles interactives complexes. La troisième dimension est principalement utilisée dans le domaine du jeu vidéo ou de la réalité virtuelle, partageant les mêmes concepts, et dans le cadre d'applications professionnelles, de type CAO, auquel cas le contenu 3D de l'application est visualisé au sein d'une fenêtre 2D.

Cependant quelques projets commencent à proposer des environnements de travail 3D avec une approche inverse : intégrer l'univers 2D dans un environnement 3D. Dans leur majorité, ils utilisent la troisième dimension comme support aux applications WIMP standard et profitent des avantages offerts par la 3D comme l'animation ou l'agencement des applications dans l'environnement virtuel, et les effets spéciaux.

Depuis plusieurs années, notre laboratoire élabore une plate-forme de travail collaboratif 3D. Aujourd'hui, nos travaux sur cette plate-forme nous amène à réfléchir à la définition d'un environnement de travail 3D générique, supportant des applications virtuelles collaboratives. Dans ce contexte, nous avons dû mettre en place une architecture logicielle pour simplifier le développement des composants interactifs qui forment les applications 3D.

Nous avons réalisé un gestionnaire de bureau 3D s'appuyant sur une adaptation du *design pattern* Modèle-Vue-Contrôleur (MVC). Nous effectuons, dans cet article, un retour d'expérience sur l'utilisation d'un modèle de ce type, dans le cadre des environnements virtuels 3D.

Dans cet article, nous commençons par décrire le concept d'application 3D au sens où nous l'entendons. Dans un second temps, après avoir présenté les travaux existant et définit les besoins d'un gestionnaire d'environnement 3D, nous présentons l'intégration dans la plate-forme du modèle MVC. Nous exposons dans la dernière partie, en nous appuyant sur des applications de tests, les apports et limitations du MVC pour la réalisation d'environnements virtuels interactifs.

NOTION D'APPLICATION 3D

Une application a pour but de permettre à l'utilisateur de réaliser des tâches. L'exécution de ces tâches est commandée par des séquences d'actions que l'utilisateur effectue dans l'interface. Dans le cadre des interfaces WIMP, l'ensemble des éléments qui forment l'application est regroupé à l'intérieur de fenêtres. Le gestionnaire d'environnement a pour rôle de gérer ces fenêtres.

Dans un environnement 3D, basé sur une représentation réaliste, une application est complètement intégrée à l'interface 3D et les éléments de l'application sont dispersés dans l'ensemble du monde virtuel. *Dans ce type d'environnement, on perd le contexte clos apporté par les fenêtres, qui encapsulent les éléments des applications. Un gestionnaire d'environnement 3D ne doit alors plus uniquement gérer chaque application globalement, mais chaque élément d'une application.*

Dans la suite de ce papier, nous prendrons Spin|3D comme exemple de gestionnaire d'environnement 3D, mais les concepts abordés sont généralisables.

Spin|3D

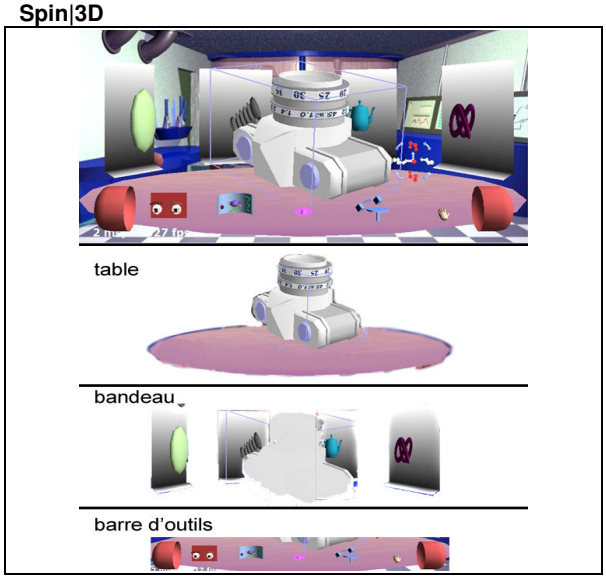

Figure 1 : Exemple d'application Spin|3D, et décomposition en ses différents éléments.

La figure 1 présente l'interface d'une application 3D développée avec la plate-forme Spin|3D. L'environnement est constitué d'une *table* centrale sur laquelle des documents 3D peuvent être placés pour être manipulés. Autour de cette *table*, un *bandeau* permet de disposer les autres documents nécessaires à l'activité. Les documents dans le *bandeau* ne sont pas directement manipulables ; ils doivent pour cela être auparavant placés sur la *table*. L'interaction entre l'utilisateur et l'interface se fait à l'aide d'*outils* qui sont activables au travers d'une barre d'outils. L'utilisateur pilote les *outils* à l'aide de deux périphériques, un périphérique servant à la désignation, un autre étant dédié à la manipulation. L'aspect collaboratif de l'application est géré par le partage d'attributs d'objets VRML97, ici l'appareil photo et ses bagues de réglages. Parmi les outils qui sont proposés à l'utilisateur, certains vont permettre d'agir sur les documents (exemple : l'outil qui sert à manipuler les bagues de l'appareil photo) et d'autres vont servir à la modification de l'environnement (exemple : l'agencement des documents dans l'interface).

Buts et contraintes

Nous nous fixons, pour la réalisation d'une plate-forme de support d'applications 3D, différents objectifs et contraintes relatifs aux comportements de l'interface :

- la plate-forme doit être suffisamment ouverte et générique pour pouvoir accueillir de nouvelles métaphores d'interaction et de représentation (par exemple bandeau ou barre d'outils...),
- les applications 3D doivent être facilement configurables pour pouvoir les adapter, éventuellement dynamiquement, à l'environnement (dispositifs physiques d'entrées/sorties, contexte de l'activité, spécificités de l'utilisateur...),
- nous devons supporter le changement de représentations dynamique des éléments de l'interface. Le besoin de changement de représentation se situe à deux niveaux : soit pour permettre une représentation différente sur les terminaux de chaque utilisateur (pour s'adapter au capacité du terminal ou de l'utilisateur du terminal), soit pour permettre, pendant le déroulement de l'activité, une représentation s'adaptant au mieux à la tâche en cours de réalisation (par exemple en repassant dans un mode d'affichage de type 2D, lorsque l'activité ou les modes d'interactions le nécessitent),
- les modalités d'interactions doivent être gérées dynamiquement. Les représentations visuelles pouvant changer en fonction du contexte matériel, il est nécessaire de pouvoir également adapter les interactions. Le besoin de changement peut également être nécessaire au cours l'activité pour s'adapter à la tâche (exemple : interdire temporairement les manipulations, ou forcer l'utilisateur à sélectionner un objet parmi un ensemble),

- toute interaction de l'utilisateur dans l'environnement 3D se fait au travers d'*outils* d'interaction.

TRAVAUX EXISTANTS

Environnements 3D

On peut distinguer plusieurs catégories d'environnements 3D :

- ceux basés sur des technologies de langage de description de graphes de scène comme X3D ou VRML97. Ces langages permettent de décrire des scènes 3D interactives. La partie dynamique de ces scènes est gérée par des langages de script. Les graphes de scène répondent assez bien au problème du placement géométrique des objets 3D, mais s'adaptent plus difficilement à du contenu réellement dynamique (i.e., nécessitant une modification de la structure du graphe de scène) et aux nouvelles techniques de rendu multi-passes nécessaire à la réalisation d'effets spéciaux sur la scène. L'interaction dans les environnements utilisant des graphes de scène se limite souvent à la manipulation d'attributs de ce graphe, comme c'est le cas pour les mondes virtuels, tels que Dive [4]. Des travaux comme [11] génèrent des applications sous forme de scènes X3D, mais leur approche consiste à effectuer un mapping des composants d'interface WIMP 2D classiques (bouton, slider...) sur une représentation 3D,
- d'autres approches, plus logicielles, utilisent l'environnement 3D comme support à l'agencement des fenêtres des applications 2D, dans un univers 3D plus ou moins interactif. Les terminaux de base, de type poste bureautique, ne possédant pas de périphériques adaptés à la navigation, des applications comme 3DNA ou Win3D offrent des déplacements semi-automatisés (des raccourcis de navigation). SphereXP limite l'espace navigable à une sphère, évitant à l'utilisateur de se perdre dans des tâches de navigation non essentielles. Ces environnements ont l'avantage d'offrir à l'utilisateur la vision simultanée d'un plus grand nombre d'informations. Rooms3D autorise les déplacements à la manière d'un jeu vidéo, répartissant les différentes activités au sein de pièces différentes. Ce genre d'environnement n'est pas adapté au travail car beaucoup trop de temps est passé dans la phase de navigation. Les applications présentées ne permettent pas d'interagir directement avec les applications 2D dans l'environnement 3D : elles repassent toutes en mode 2D classique pour l'interaction. Enfin, elles restent très proches des concepts d'IHM 2D,
- Muse et le Workspace3D [10] proposent des environnements 3D pour le travail collaboratif. L'univers 3D contient des applications qui vont pouvoir être utilisées à plusieurs. L'avantage apporté par la troisième dimension, dans ce cadre, est la possibilité de visualiser les actions effectuées par les autres acteurs. Les applications 2D (navigateur Internet, ou Microsoft PowerPoint, par exemple) sont mieux intégrées dans ces environnements : les utilisateurs, par navigation semi-automatique, se placent face au *mur* supportant le document 2D lorsqu'ils désirent interagir avec celui-ci. Le passage de ce document en mode 2D se fait alors naturellement, sans *couture*,
- les récents travaux de Sun Microsystems sur le projet Looking Glass [**Erreur ! Source du renvoi introuvable.**9], ou la plate-forme XYZ [6], partent eux aussi du principe d'intégration d'applications 2D existantes dans l'environnement. Ici, les applications 2D sont également plaquées sur des boites 3D, mais les documents 2D sont de plus directement manipulables en 3D. D'autre part, Looking Glass, via une API en Java, et XYZ via l'API XYZ, offrent la possibilité de créer des applications entièrement 3D. Cette approche semble la plus prometteuse, au sens où elle permet l'utilisation d'applications 2D existantes, mais s'ouvre également au développement de nouvelles applications entièrement 3D. Ce type d'environnement permet une migration plus douce entre les mondes 2D et 3D.

Architectures de systèmes interactifs

Les architectures Seeheim et ARCH [1] sont des modèles conceptuels qui permettent de découper l'application en plusieurs couches allant du noyau fonctionnel à la présentation. Elles offrent l'avantage de séparer l'interface du noyau fonctionnel de l'application d'avec les mécanismes de représentation et d'interaction du système. Ces architectures nécessitent d'avoir un noyau central regroupant l'ensemble des fonctionnalités, et, de ce fait, s'adaptent assez mal au cadre d'une plate-forme générique.

L'architecture PAC [5] est composée d'agents interactifs, chaque agent étant formé de 3 parties : la Présentation qui gère la représentation visuelle et l'interaction de l'agent, l'Abstraction qui contient le code métier de l'agent, et le Contrôle qui définit la voie de communication entre la présentation et l'abstraction. Dans le paradigme PAC, les agents sont organisés de manière hiérarchique, permettant ainsi de décrire le système interactif de manière progressive. PAC-Amodeus [8] est une architecture hybride, où le Contrôleur de dialogues de Arch est composé d'une hiérarchie d'agent PAC. PAC-Amodeus apporte une meilleure portabilité.

Le modèle DPI [2] est une architecture composant centrée autour de la métaphore du document. Le Document décrit par l'ensemble de ses propriétés, la Présentation est chargée de la représentation et l'interaction s'effectue au travers d'Instruments.L'utilisation de la métaphore d'instruments permet de pouvoir utiliser des instruments réalisant des actions génériques sur des documents différents. Néanmoins, le résultat de la consommation d'une action par un document est figé au niveau du document.

Le modèle MVC, issu de Smalltalk [7], décrit un système d'agents à trois facettes : le Modèle contenant le noyau fonctionnel de l'agent, la Vue chargée de la représentation du Modèle, et le Contrôleur gérant ce qui a trait à l'interaction avec l'utilisateur. Ce design pattern est le plus connu, largement utilisé, et l'on en trouve beaucoup d'adaptations : dans Swing, par exemple, où Vue et Contrôleur ont été fusionnés. Les agents peuvent également être organisés de manière hiérarchique, afin de mieux s'adapter au système interactif. Le principal inconvénient du paradigme MVC reste la grande multiplicité de classes à implémenter.

VERS UN GESTIONNAIRE D'ENVIRONNEMENT 3D

La 3D apporte des avantages en terme de représentations, en offrant la possibilité d'afficher un plus grand nombre d'informations [3]. Dans le cadre d'une activité de travail, nous pensons que la navigation est à éviter, qu'il faille utiliser les atouts de la 3D pour l'agencement des documents, mais surtout, qu'il faut introduire de nouveaux concepts pour les IHM 3D. Pour certaines applications provenant du monde 2D, le passage à la troisième dimension ne se justifie que pour l'agencement des fenêtres dans l'espace. Par contre, traduire tous les composants interactifs 2D (boutons, barre de défilement) par leurs homologues 3D n'apporte que très peu d'intérêts, mis à part esthétique. L'effort demandé à l'utilisateur pour interagir avec ces composants devient plus important. Par contre, un gestionnaire d'environnement virtuel 3D doit permettre de définir des nouvelles modalités d'interaction et introduire de nouveaux contrôles 3D, au travers de nouvelles métaphores, n'ayant pas leur équivalent 2D.

Les approches logicielles offrent une plus grande liberté en terme de contrôle et de performance que l'utilisation de format de description standard comme X3D (qui disposent d'outils auteur adaptés), mais au détriment d'un temps de développement plus élevé des applications. L'idéal est donc de pouvoir hybrider les deux approches, en offrant la possibilité de décrire une partie de l'application par un langage de description, mais également de permettre la définition de composants programmés dans des langages évolués utilisant directement une API du système.

Définitions

Dans cette section nous explicitons les concepts et le vocabulaire correspondant à notre définition d'un Bureau 3D. Certaines définitions sont spécifiques aux environnements 3D, d'autres sont des extensions de leur équivalent 2D :

- bureau et application 3D : un bureau 3D est un environnement de travail 3D non immersif gérant des applications 3D. Une application 3D est composée d'objets 3D réagissant aux actions déclenchées par l'utilisateur. L'utilisateur agit sur l'environnement à l'aide d'outils d'interaction,
- outil d'interaction : objet 3D de l'environnement piloté par l'utilisateur au travers des périphériques. Les outils ont pour rôle d'agir sur les documents présents dans l'interface. Un outil possède un comportement par défaut, indépendant du contexte courant.
- Zone d'interaction : une zone est un volume de l'espace virtuel possédant une représentation et supportant une modalité d'interaction spécifique. Par exemple, dans l'application de la Figure 1, le *bandeau* est une zone d'interaction où la manipulation d'objet n'est pas autorisée,
- modalité d'interaction d'une zone : possibilité de modifier dynamiquement le comportement par défaut d'un outil, pour interdire, augmenter ou substituer l'action de cet outil,
- contexte d'interaction : définit l'état interactif du système et relate l'activité de l'utilisateur sur le système à un moment donné. Il contient comme information : l'outil que l'utilisateur manipule, la zone dans laquelle se trouve l'outil, l'action que l'outil est en train d'effectuer, le document manipulé, l'état des périphériques...

Etude de l'utilisation du modèle MVC

Dans le contexte d'environnement 3D tel que nous le définissons, nous avons retenu le paradigme MVC comme modèle de conception. Il nous a semblé le plus adapté car chaque composant interactif ou fonctionnel de l'environnement peut se décrire sous la forme d'une triade MVC, l'ensemble de ces composants formant l'application fonctionnelle. Notre but est de voire jusqu'à quel point ce modèle s'adapte à la réalisation d'interfaces 3D interactives configurables, en l'utilisant pour implémentation d'une application 3D. L'étude porte également sur comment étendre ce modèle pour qu'il puisse répondre aux deux principales contraintes que nous nous sommes fixées : avoir une configurabilité totale de l'application et un mécanisme d'interaction adaptable au contexte

Le MVC ne force pas l'utilisation d'un noyau regroupant toute l'application, et permet la répartition du code métier dans tous les objets contenus dans l'environnement. Contrairement à l'approche d'Okada [12] nous conservons un découpage fort des composants MVC qui permet de dégager des comportements d'interaction réutilisables sur différentes vues. L'interaction quant à elle s'effectue en partie via des outils d'interaction (assimilable au concept d'instrument) et n'est pas uniquement gérée par le Contrôleur.

IMPLEMENTATION

Description du modèle MVC pour les zones

Une zone d'interaction est implémentée à l'aide d'une triade MVC. A une zone, nous associons :

- un Modèle, offrant les fonctionnalités associées à la zone,
- des Contrôleurs, responsables des modalités d'interaction associées à la zone,
- des Vues, responsables de la visualisation de la zone.

Pour permettre un maximum de ré-utilisabilité, les modèles, vues et contrôleurs sont implémentés sous forme de modules dynamiques (dll/dso) indépendants, codés en C++ ou Javascript. Ces modules sont chargés dynamiquement à l'exécution et la manière de les associer pour définir les zones d'interactions est décrite dans un fichier de configuration au format XML. L'exemple suivant décrit la configuration de la zone associée à la *table* (figure 1). On y retrouve la définition du modèle, les différentes vues de la zone, et les contrôleurs (ici, chargés d'activer/désactiver le mode de transparence des documents sur la table en fonction de la position du pointeur 3D). La syntaxe reste suffisamment ouverte pour pouvoir spécifier les paramètres nécessaires aux différents modules.

```
<zone id="Table">
  <model id="TabM" instance="GenericModel.dll" />
  <views active="TabV1">
    <view id="TabV1" instance="TableView.dll">
      <params>
        <param name="texture" value="Bluetile.jpg" />
      </params>
    </view>
  </views>
  <controllers>
    <controller id="EnTrans" instance="TranspC.dll">
      <params>
        <param name="enabled" value="TRUE" />
      </params>
    </controller>
    <controller id="DisTrans" instance="TranspC.dll">
      <params>
        <param name="enabled" value="FALSE" />
      </params>
    </controller>
  </controllers>
</zone>
```

Code 1 : *Exemple de fichier de configuration d'une zone.*

Une zone peut posséder plusieurs vues, mais une seule vue est active à un instant donné. La vue active peut être choisie dynamiquement durant l'exécution. Les vues sont connectées au modèle et entretiennent avec lui une relation observable/observeur. Cette relation va permettre la notification automatique de la vue de tout changement d'état du modèle.

L'environnement 3D complet est constitué d'un ensemble de zones organisées de manière hiérarchique. Cette hiérarchie permet de donner à une zone la possibilité de contrôler sa sous-hiérarchie. Une zone particulière correspondant à la racine de la hiérarchie (*SystemArea*, dans l'exemple ci-dessous) représente l'intégralité de l'interface.

```
<Workspace>
  <zone id=SystemArea>
    <zone id=Bandeau>
      <object name=document.wrl />
    </zone>
    <zone id=Table>
      <object name=object.wrl />
    </zone>
    <zone id=ToolBar>
      <zone id=ExchangeTool />
      <zone id=SelectTool />
    </zone>
  </zone>
</Workspace>
```

Code 2 : *Exemple de fichier de configuration du graphe de zones de l'application.*

Le modèle

Le modèle correspond au code métier de la zone, tout modèle hérite d'une classe C++ ***Model*****,** offrant un minimum de fonctionnalités :

- la gestion d'attributs dynamiques : un modèle va pouvoir posséder des attributs typés accessibles par les autres modules. Ils peuvent être créés par définition dans le fichier de configuration, ou dynamiquement par programmation. Le modèle sert dans ce dernier cas uniquement de mémoire de stockage partagée, sans connaissance de l'utilité de ces attributs. La déclaration d'attributs sert, par exemple, à pouvoir faire communiquer des vues et des contrôleurs sans avoir à implémenter un modèle dédié,
- la gestion de la sous-hiérarchie : c'est le modèle qui contient les sous-zones et les objets contenus dans la zone. En mettant cette gestion à ce niveau, nous permettons au modèle d'avoir la main mise sur la modification du contenu de la zone, puisque tout passe par lui.

La vue

La vue est responsable de l'affichage de la zone, et du déclenchement de l'affichage des sous-zones et des objets contenus dans le modèle. Pour afficher l'environnement 3D complet, le noyau de la plate-forme demande uniquement à la zone racine de s'afficher. Le rendu de la hiérarchie complète se fait alors de proche en proche, chaque noeud affiché initiant l'affichage des noeuds suivants. Une vue a ainsi la maîtrise totale de l'affichage de la zone qu'elle représente et peut contrôler l'affichage des sous-zones.

La vue fournit les mécanismes de détection de collisions entre un rayon ou un point de l'espace et ce qu'elle affiche. En effet, comme c'est la seule à connaître le placement des objets dans l'espace 3D, elle est la seule à pouvoir renseigner ce type d'informations. Ces informations lui sont demandées par la plate-forme pour déterminer le contexte d'interaction.

Le contrôleur

Le contrôleur a pour rôle la définition de la modalité d'interaction spécifique d'une zone et la modification de l'état du modèle en conséquence. La particularité de notre implémentation du modèle MVC vient de la manière avec laquelle le contrôleur invoqué : les contrôleurs sont liés aux outils d'interaction par un gestionnaire d'interactions dont la description sort du cadre de cet article ; on retiendra uniquement qu'un contrôleur de zone spécifie une action déclenchée pour interdire, étendre ou substituer la modalité d'interaction par défaut d'un outil. Dans notre approche, le contrôleur pourra être invoqué pour modifier le comportement défini par l'outil d'interaction. Le type d'adaptation est régi à l'aide de règles d'interaction décrites en xml spécifiant si le contrôleur doit se substituer à l'outil, l'augmenter ou l'interdire.

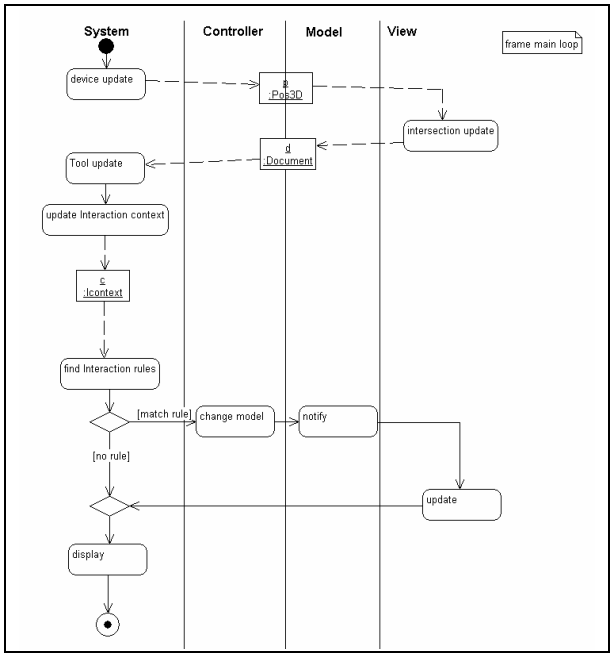

Figure 2 : Diagramme d'activité pour une boucle d'affichage.

Mise en relation des modules

La figure 2 reprend le diagramme d'activité simplifié du déroulement d'une boucle d'affichage :

- le système récupère les informations provenant des différents périphériques, et effectue la détection de collisions avec la scène,
- les vues remplissent le contexte d'interaction en spécifiant dans quelles zones se trouve l'outil courant, le document désigné, etc…,
- les informations contenues dans le contexte d'interaction sont utilisées par l'outil d'interaction pour agir sur l'environnement. Le gestionnaire d'interaction peut éventuellement modifier l'action par défaut de l'outil, en invoquant les contrôleurs des zones concernées,
- les vues associées aux modèles modifiés sont alors mises à jour.

Diagramme de classes

La figure 3 reprend le diagramme de classes du composant MVC utilisé pour définir une zone d'interaction : le modèle entretient avec la vue une relation *observeur/observable*, le contrôleur agit sur le modèle, et un modèle définit la hiérarchie de l'environnement 3D en contenant plusieurs documents et sous-composants MVC.

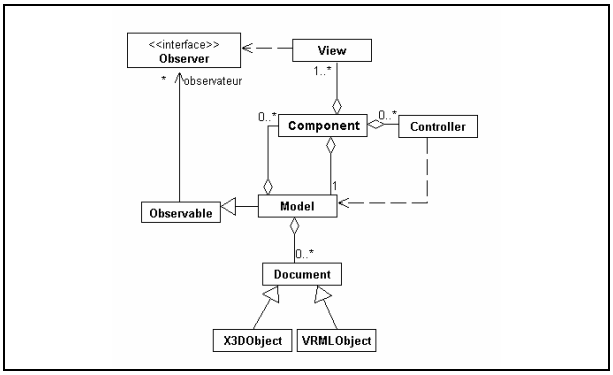

Figure 3 : Diagramme UML du composant MVC.

TEST ET VALIDATION DU MODELE

Dans cette section, nous présentons trois applications ayant servi à valider notre utilisation du modèle MVC pour la réalisation d'applications configurables

Modification dynamique de la modalité d'interaction

Le premier exemple décrit une application s'appuyant sur la métaphore de la table de réunion (figure 1). Elle permet de mettre en évidence l'adaptation de l'interaction en fonction des zones. Cette application est composée de 4 zones:

- une table représentant l'espace de manipulation (de travail),
- un bandeau entourant la table et contenant les documents disponibles pour l'activité en cours,
- une zone contenant une barre d'outils, et permettant de masquer/afficher cette barre d'outils.
- une barre horizontale contenant des objets 3D représentant les différents outils disponibles.

Hormis les icônes des outils, tous les éléments de l'interface utilisent le même *Modèle* générique, à savoir un conteneur de documents possédant une interface d'attributs modifiables dynamiquement. Ce constat nous amène à penser que peu de zones ont besoin d'un code fonctionnel propre. Dans la majorité des cas, il s'agit uniquement de partage d'attributs simples entre *Vues* et *Contrôleurs*.

Nous avons également pu définir deux catégories de *Contrôleurs* : ceux spécifiques à une représentation, c'est à dire étroitement liés à une *Vue*, et ceux définissant un comportement d'interaction propre aux documents contenus dans la zone. Dans notre application d'exemple, la zone du bandeau interdit toute manipulation des documents qu'elle contient. Un *Contrôleur* attaché à cette zone aura donc pour rôle d'empêcher les actions de sélection effectuées par l'utilisateur et d'afficher un feedback visuel servant à informer de cet échec (halo rouge clignotant autour du document qui était sur le point d'être sélectionné). Ce *Contrôleur* est propre au comportement d'interaction spécifique du bandeau, mais n'est en aucun cas lié à la représentation de ce dernier. A l'inverse, un *Contrôleur* permettant d'orienter le bandeau sous l'action du périphérique de manipulation est lui lié à la *Vue* : il modifie un attribut d'orientation ajouté par configuration au *Modèle*, attribut qui sera ensuite utilisé par la *Vue* pour son affichage.

La barre d'outil est également réalisée à l'aide d'un composant MVC, et d'un *Contrôleur* déclenchant le changement d'outil courant. Il n'est ainsi pas besoin, contrairement au cas des environnements 2D, d'un mécanisme spécifique de déclaration d'une barre d'outils ou de menus déroulant.

Réutilisabilité des composants

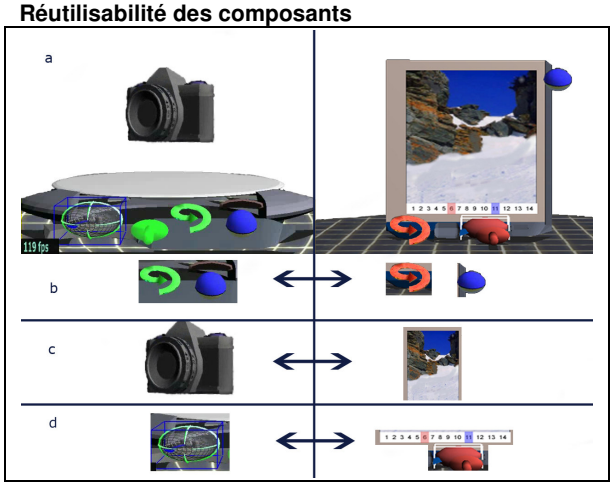

Figure 4 : La métaphore de prise de point de vue: à gauche, la version pour les documents 3D, à droite pour les documents 2D (ici, une liste d'images). a) outil complet. b) outil de prise de vue distant et retour. c) les documents. d) contrôle du point de vue local et représentation des points de vues distants.

La seconde application (figure 4), qui met en oeuvre une métaphore de prise de point de vue est plus représentative du potentiel du système en terme de modularité. La métaphore de prise de point de vue permet de naviguer dans un document et de pouvoir acquérir le point de vue des utilisateurs distant sur ce document. Les documents supportés sont soit des objets 3D (auquel cas la navigation est assimilée à l'orientation de l'objet), soit des documents 2D comme des diaporama ou des documents texte multi-pages (auquel cas la navigation correspond à un changement de page). La manipulation du document se fait par interaction indirecte (via une *trackball* virtuelle pour les objets 3D, via une molette de défilement pour les documents 2D : figure 4.d).

La métaphore a été découpée en trois parties :

1. une zone d'interaction servant à la manipulation et l'affichage des différents points de vues (figure 4.d),
2. une zone supportant l'affichage et la manipulation du document 2D ou 3D (figure 4.c),
3. une zone permettant de synchroniser/désynchroniser son point de vue avec celui d'un autre utilisateur (figure 4.b).

L'architecture MVC nous a permis de réduire le coût de développement des deux versions de la métaphore : il est en effet assez naturel de remarquer que seule la présentation visuelle diffère, en fonction du type du document. Le noyau fonctionnel consiste, dans les deux cas, à pouvoir manipuler des attributs représentant un point de vue (orientation ou numéro de page), et à pouvoir récupérer les attributs d'orientation d'un utilisateur distant.

Dynamicité de la représentation

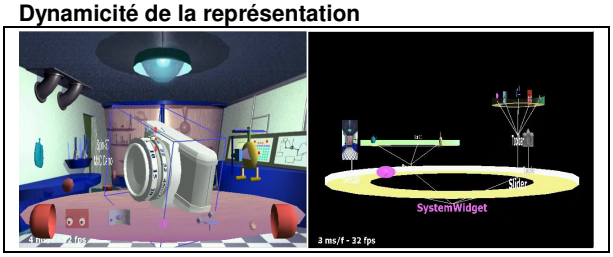

Figure 5 : à gauche l'application en mode standard, à droite, les vues de chaque zone ont été remplacées par une vue cone-tree.

Dans le cadre de l'évolution d'une activité, il peut être nécessaire de modifier la représentation de l'environnement pour l'adapter à la tâche sans changer la sémantique de l'application. Ce dernier exemple met en évidence l'intérêt de la définition d'une classe ***Model*** de base, dont hérite tout *modèle* des composants MVC. Nous avons implémenté une vue visualisant le contenu d'une zone sous forme de cone-tree. Cette vue utilise directement l'interface de la classe ***Model*** de base, et peut donc être attachée à n'importe quelle zone. La figure 5 montre le résultat de l'affichage : la même scène est affichée avec des vues différentes. Les modalités d'interaction restent les mêmes entre les deux représentations : la sélection qui n'était pas autorisée dans le bandeau dans le mode standard, reste proscrite pour le sous-arbre correspondant à la zone du bandeau dans le mode cone-tree.

RESULTATS APPORTS ET LIMITATIONS

L'utilisation du modèle MVC a permis de répondre aux besoins émis par les applications 3D. L'approche modulaire apportée et la définition de composants MVC par configuration XML permet de faciliter le prototypage au niveau des représentations et de l'interaction.
Par contre, les modules ont besoin d'être finement documentés et exempts de bogues, c'est une condition nécessaire à la réutilisation. Le nombre de modules peut également rapidement augmenter dans le cadre d'une application complexe, et le choix du découpage en modules peut ne pas être évident. C'est ce que le modèle PAC-Amodeus tente de régler en partie, grâce à l'utilisation de différentes heuristiques de conception. Ce type d'heuristiques ne peut cependant être défini qu'en effectuant une analyse sur les choix d'implémentation de plusieurs applications suffisamment complexes.

CONCLUSIONS ET TRAVAUX FUTURS

L'utilisation du modèle MVC tel qu'il a été présenté dans cet article a révélé de nombreux avantages. Le développement d'une zone d'interaction est assez simple et permet de prototyper/tester assez rapidement de nouveaux concepts. La structure hiérarchique de l'application s'adapte bien au rendu 3D en analogie avec les graphes de scènes classiques. Des zones d'agencements peuvent être imaginées pour structurer la représentation de l'application. Le fait que les applications soient entièrement décrites dans un fichier de configuration de manière modulaire permet de construire les applications progressivement et d'isoler rapidement les sources de bogues.

Les changements dynamiques des *Vues*, permettant de modifier, en cours d'exécution, la représentation d'une application tout en conservant les modalités d'interaction, sont un des apports majeurs de l'architecture. Cependant, l'application conserve la même structure de composition hiérarchique des zones. Il faudrait en plus des changement de *Vues* pouvoir remanier le graphe de composants MVC pour pouvoir obtenir, le cas échéant, une application sémantiquement différente. Cette fonctionnalité est pour l'instant réalisable par programmation et n'est pas configurable.

Les résultats que nous avons obtenu sont plutôt positifs et nous font nous pencher sur l'extension du modèle MVC aux documents même de l'application. Ces documents sont, pour l'instant, soit des objets 3D au format VRML97, soit des composants applicatifs de type boites noires. L'utilisation du modèle MVC sur les documents permettrait d'uniformiser la plate-forme et d'apporter les avantages de l'architecture au niveau de la description des documents dont l'adaptation des modalités d'interactions.

REMERCIEMENTS

Ces travaux ont été menés dans le cadre du projet Alcove et sont soutenus par l'Ircica (Institut de Recherche sur les Composants logiciels et matériels pour l'Information et la Communication Avancée) et FranceTélécom R&D.

BIBLIOGRAPHIE

1. "A Metamodel for the Runtime Architecture of an Interactive System", The UIMS Tool Developers Workshop, *SIGCHI Bulletin*, ACM, 24, 1, 1992, pp. 32-37.
2. Beaudoux, O. et Beaudouin-Lafon M. DPI: A Conceptual Model Based on Documents and Interaction Instruments. *People and Computer XV - Interaction without frontier (Joint proceedings of HCI 2001 and IHM 2001, Lille, France),* Spring Verlag, pp. 247-263.
3. Card, S.K., Robertson, G.G. et York, W. The WebBook and the WebForager: an information workspace for the World Wide Web, *CHI 96, ACM Conference on Human Factors in Software*, ACM Press, New York. pp. 111-117.
4. Carlsson, C. et Hagsand, O. – "DIVE – A Multi User Virtual Reality System", Actes de IEEE Virtual Reality Annual *International Symposium (VRAIS)*, Septembre 1993, pp. 394–400.
5. Coutaz, J. PAC: an Implementation Model for Dialog Design, *Proceedings of Interact'87*, H-J. Bullinger, B. Shackel ed., North Holland, Stuttgart, (Sept. 1987), pp. 431-436.
6. Knispel, J. The XYZ Virtual Workspace, *Proceeding IEEE Virtual Reality*, 2005, pp.241-245
7. Krasner, G.et Pope, S. A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80, *Journal of Object Oriented Programming*, (Août/Sept. 1988), pp. 26-49.
8. Nigay, L. " Conception et modélisation logicielles des systèmes interactifs : application aux interfaces multimodales". Thèse de doctorat de l'Université Joseph Fourier, 1994.
9. Project Looking Glass, http://www.sun.com/software/looking_glass/
10. Tixeo WorkSpace3D, http://www.tixeo.com/
11. Vanderdonckt, J., Chieu, C. K., Bouillon, L. et Trevisan, D. Model-based design, generation, and evaluation of virtual user interfaces. *Web3D '04: Proceedings of the ninth international conference on 3D Web technology*, ACM Press, Monterey, California, 2004, pp. 51-60
12. Yoshihiro Okada, 3D visual component based approach for immersive collaborative virtual environments. *ETP '03: Proceedings of the 2003 ACM SIGMM workshop on Experiential telepresence*. Berkeley, California, ACM Press, pp 84-90